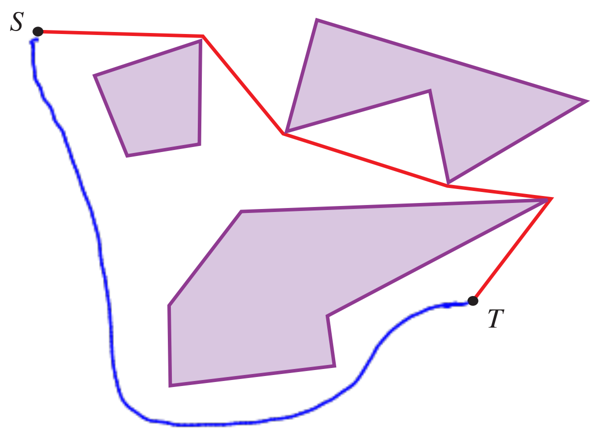
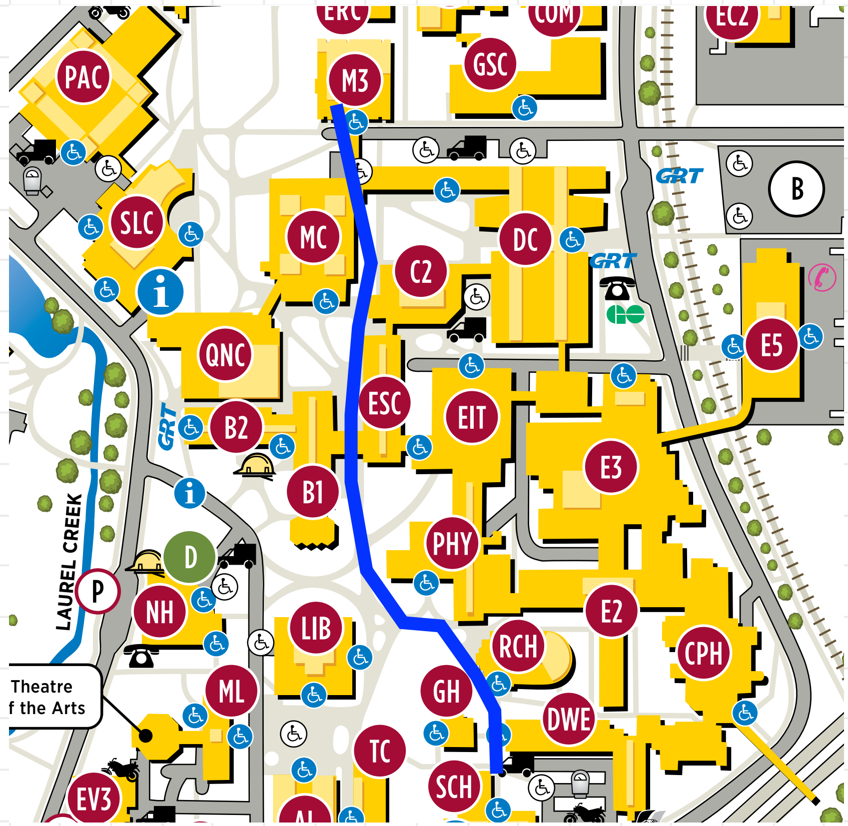
Shortest paths in the plane with polygonal obstacles

Given some polygons ("obstacles") in the plane, a start point s and end point t, find the shortest path from s to t that avoids the obstacles.



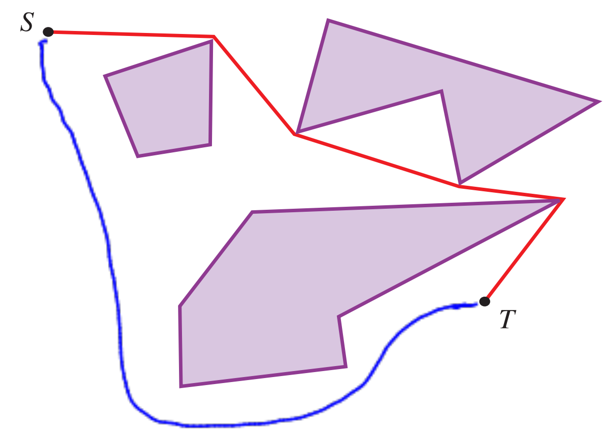


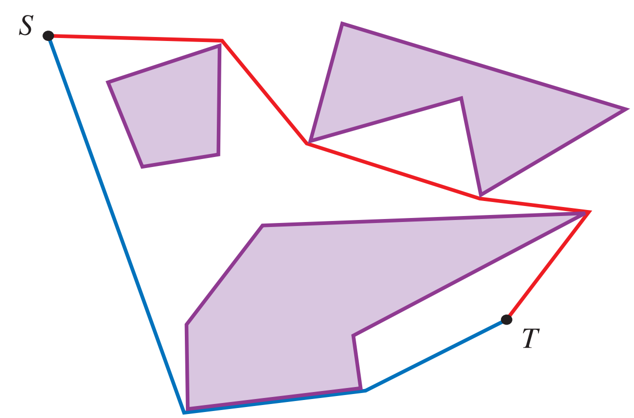
Note: most solutions actually allow us to find the shortest path from s to every t ("single source" shortest path problem).

Shortest paths in the plane with polygonal obstacles

Lemma. Any shortest path among obstacles in the plane is composed of line segments between vertices of the obstacles. Also any ***locally shortest*** path.

locally shortest
= no local change can shorten the path
= taut string solution

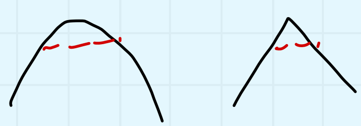




2 locally shortest paths from S to T

Proof.

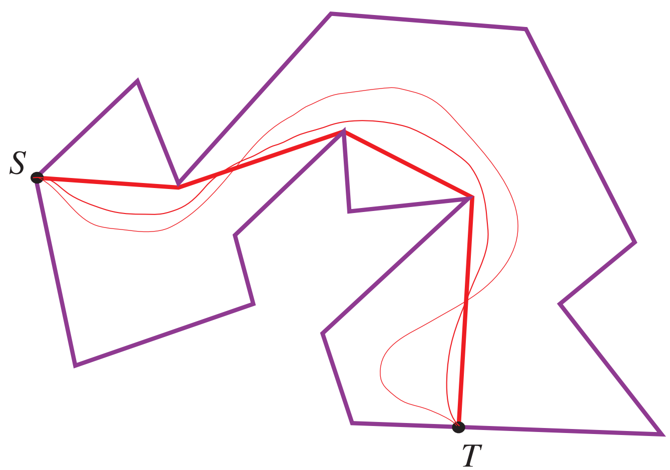
Any curve or bend not at a vertex can be shortened.



Shortest paths in a simple polygon

In a simple polygon, there is only one locally shortest path from S to T.

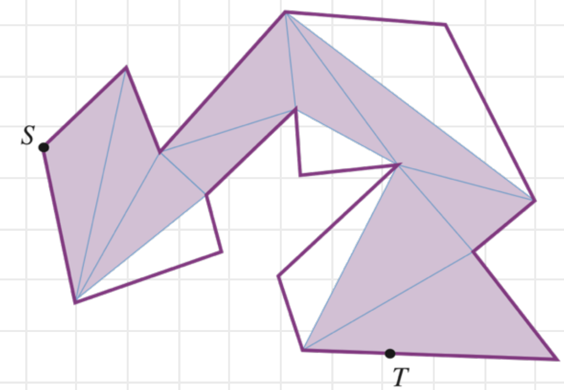
Can be found in O(n) time, after triangulating the polygon. “Funnel algorithm”.
(Recall: triangulation in O(n) time by Chazelle’s algorithm.)



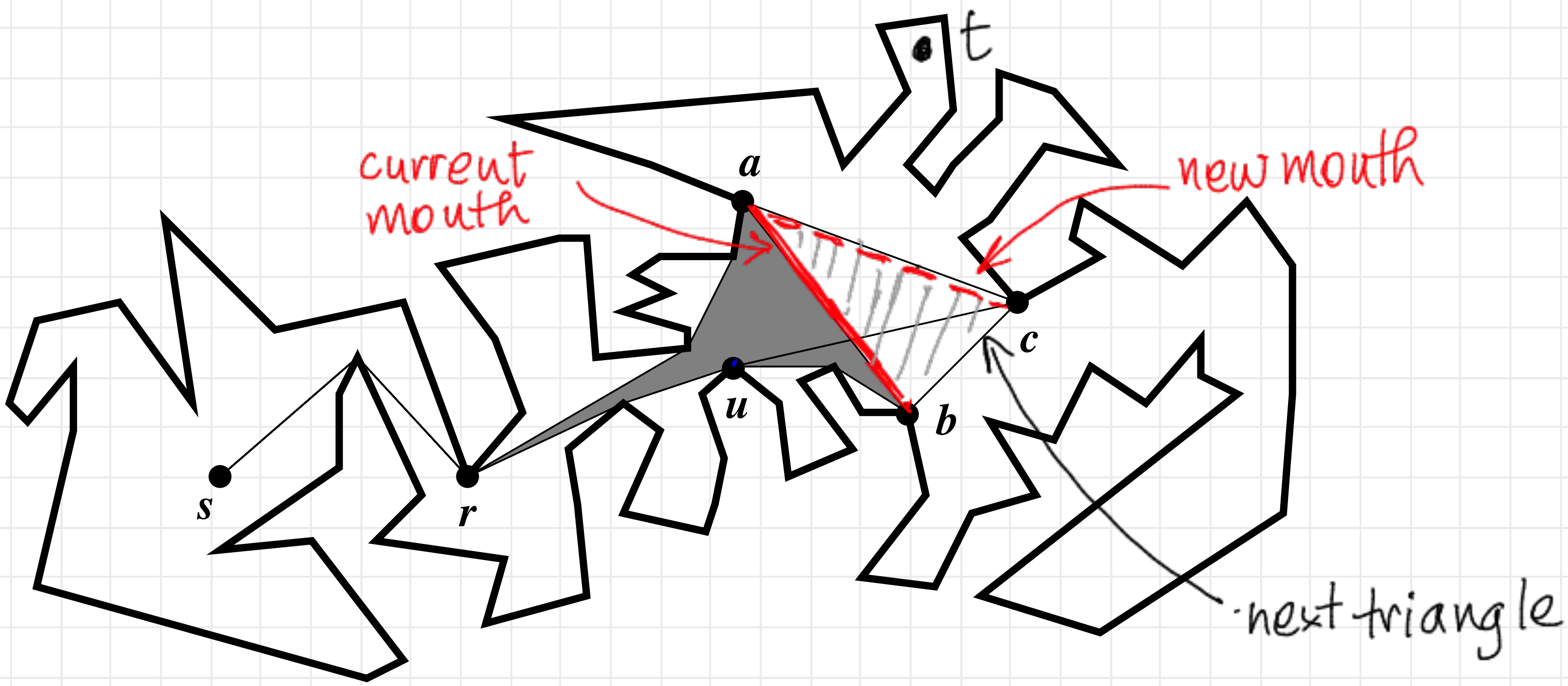
OPEN: can this be done in linear time without Chazelle’s algorithm?

Shortest paths in a simple polygon — the funnel algorithm

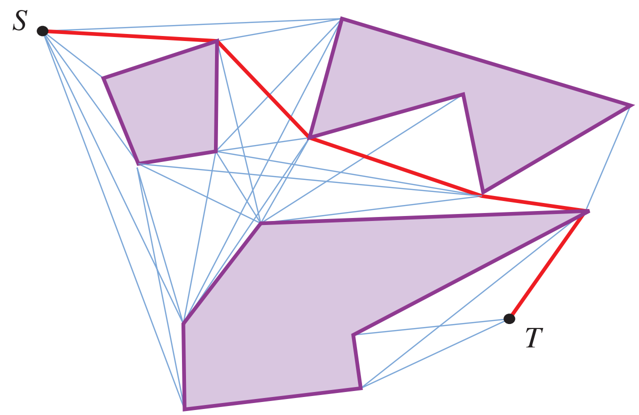
- triangulate the polygon and find the path of triangles from s to t



- go along the path of triangles, maintaining shortest paths from s to the ***mouth*** of the current triangle



Shortest paths in the plane with polygonal obstacles



Two main approaches:

1. find a shortest path in the **visibility graph** using Dijkstra's shortest path algorithm. $O(n^2)$ because the graph may have many edges.

2. "continuous" Dijkstra approach

Note: real-RAM model of computation, since we compare sums of square roots.

Shortest paths in the plane with polygonal obstacles via Visibility

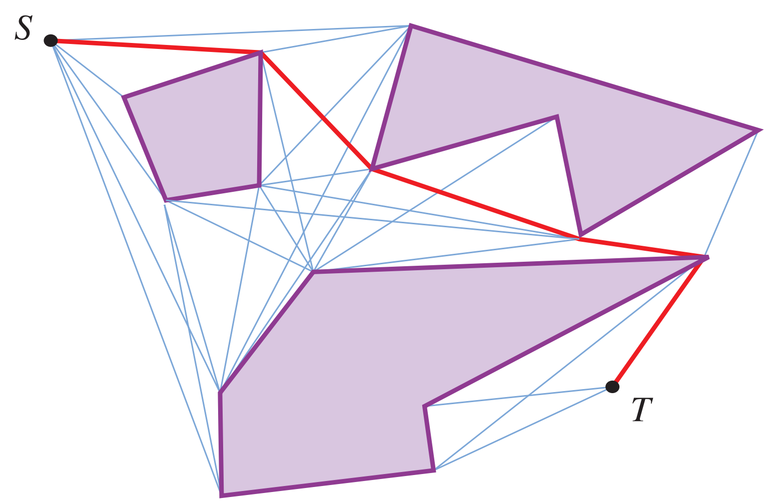
Visibility graph:

Nodes are vertices of the polygonal obstacles plus S and T.
Edge (a,b) if the line segment ab does not intersect the interior of any obstacle.
weight (a,b) = Euclidean length of segment ab.

Problem becomes: find the shortest path from S to T in the visibility graph.

Use Dijkstra's shortest path algorithm.
Run time O(m + n log n)
m = #edges.

But m can be Theta(n^2).



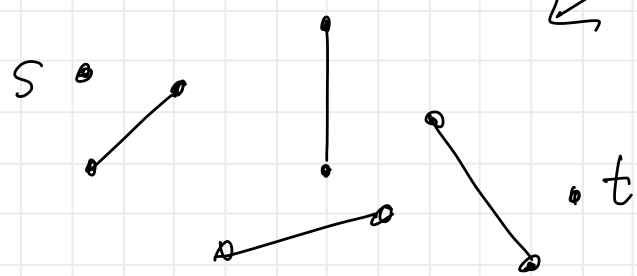
Shortest paths in the plane with polygonal obstacles via Visibility

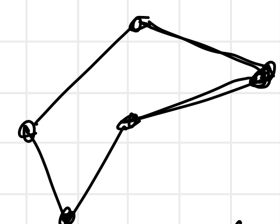
Computing the visibility graph.

Obvious: $O(n^3)$
Plane sweep: $O(n^2 \log n)$
Line arrangements: $O(n^2)$

Output sensitive: $O(n \log n + k)$, k = output size = number of edges of vis. graph. [Ghosh and Mount, 1991].
Huge efforts went into this line of research, but the bottleneck is that the visibility graph can have n^2 edges.

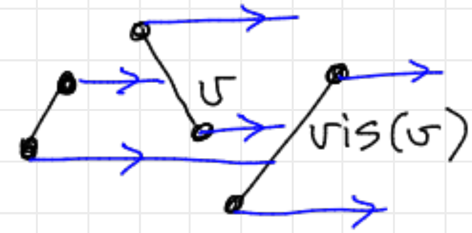
Next: the $O(n^2)$ algorithm via line arrangements [Welzl, 1985]
(described for non-degenerate disjoint line segments).





here (with polygonal obstacles) the line segments meet at point.

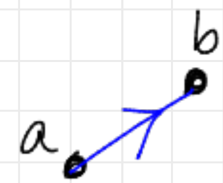
Computing the visibility graph in $O(n^2)$. [Welzl, 1985]
(described for **non-degenerate disjoint** line segments).



1. shoot a horizontal ray from every vertex to find which segment it sees to the right. vis(v) = segment it sees. O(n log n) via plane sweep. horiz. line top to bottom.

2. sweep the direction vector cyclically, maintaining vis(v)

we will find a visibility edge (a,b) when the direction vector is parallel to it.



How to sweep the direction vector:

Visibilities only change when the direction vector goes through 2 vertices.

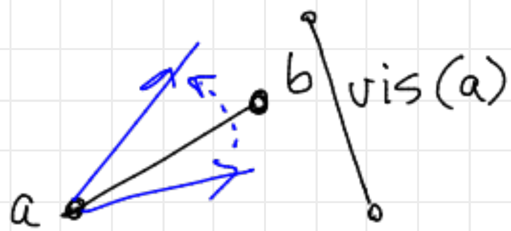
Find all (n choose 2) lines through pairs of points and sort by slope. $O(n^2 \log n)$
we will see a faster way using arrangements!

Computing the visibility graph in O(n^2).

How to update vis(v) for the line through points a, b.
Note: only vis(a) and vis(b) can change. Update vis(a) in O(1):

and output visibility edges

case 1. ab is a segment

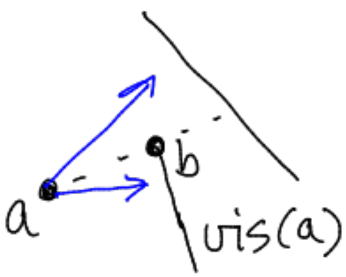


vis(a) is same
output (a,b)

case 2. a does not see b
then vis(a) blocks a from seeing b

same vis(a)

case 3. b is an endpoint of vis(a)



vis(a) ← vis(b)
output (a,b)

case 4. otherwise

vis(a) ← segment(b)
output (a,b)

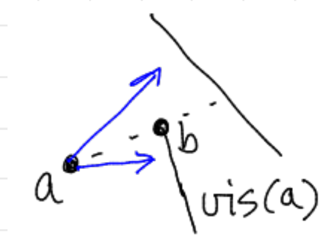
Computing the visibility graph in $O(n^2)$.

Each update is $O(1)$. Total cost of updates is $O(n^2)$.

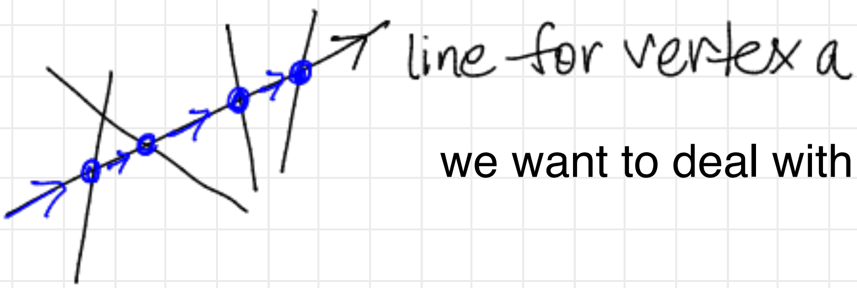
The bottleneck is $O(n^2 \log n)$ to sort the slopes. Do we need to do that?
NO. We just need that the n-1 lines through any one point a must be handled in the correct order.

Case 3 is crucial — we need vis(b) to be correct when we handle direction ab. But that's ok if we have the correct order around b.

case 3. b is an endpoint of vis(a)



Plan. Take the dual.
Vertices (points) become lines.
Directions (lines through 2 points) becomes points.

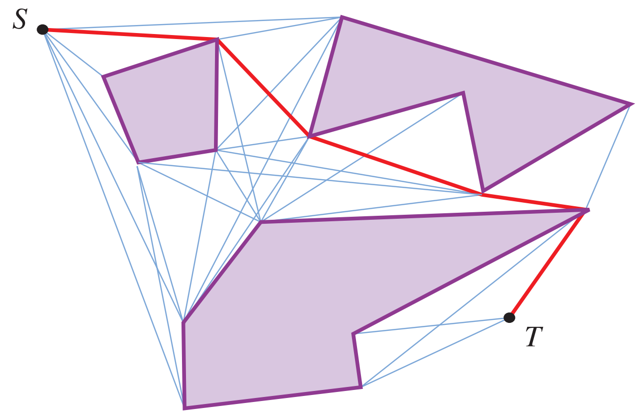


we want to deal with the points on line a in order.

Compute the arrangement. $O(n^2)$. Direct edges left to right.
This gives a directed acyclic graph. Now use a "topological order" of the graph.
This avoids sorting.

Recall

Shortest paths in the plane with polygonal obstacles



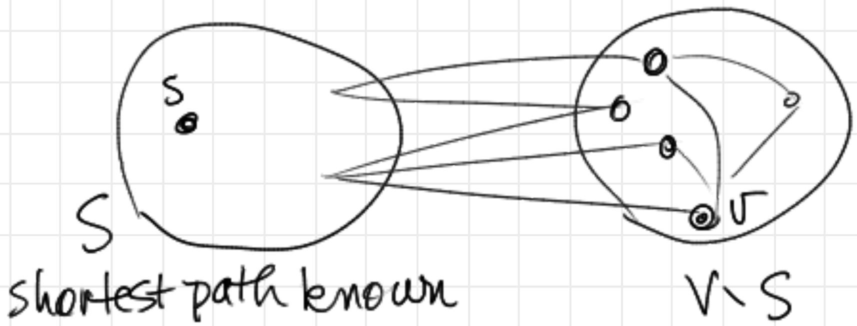
Two main approaches:

1. find a shortest path in the **visibility graph** using Dijkstra's shortest path algorithm. $O(n^2)$ because the graph may have many edges.

2. "continuous" Dijkstra approach

Note: real-RAM model of computation, since we compare sums of square roots.

Reminder of Dijkstra's algorithm
single source shortest paths for non-negative edge weights.



d(v) = shortest path from s to v
using vertices in S plus one edge to v

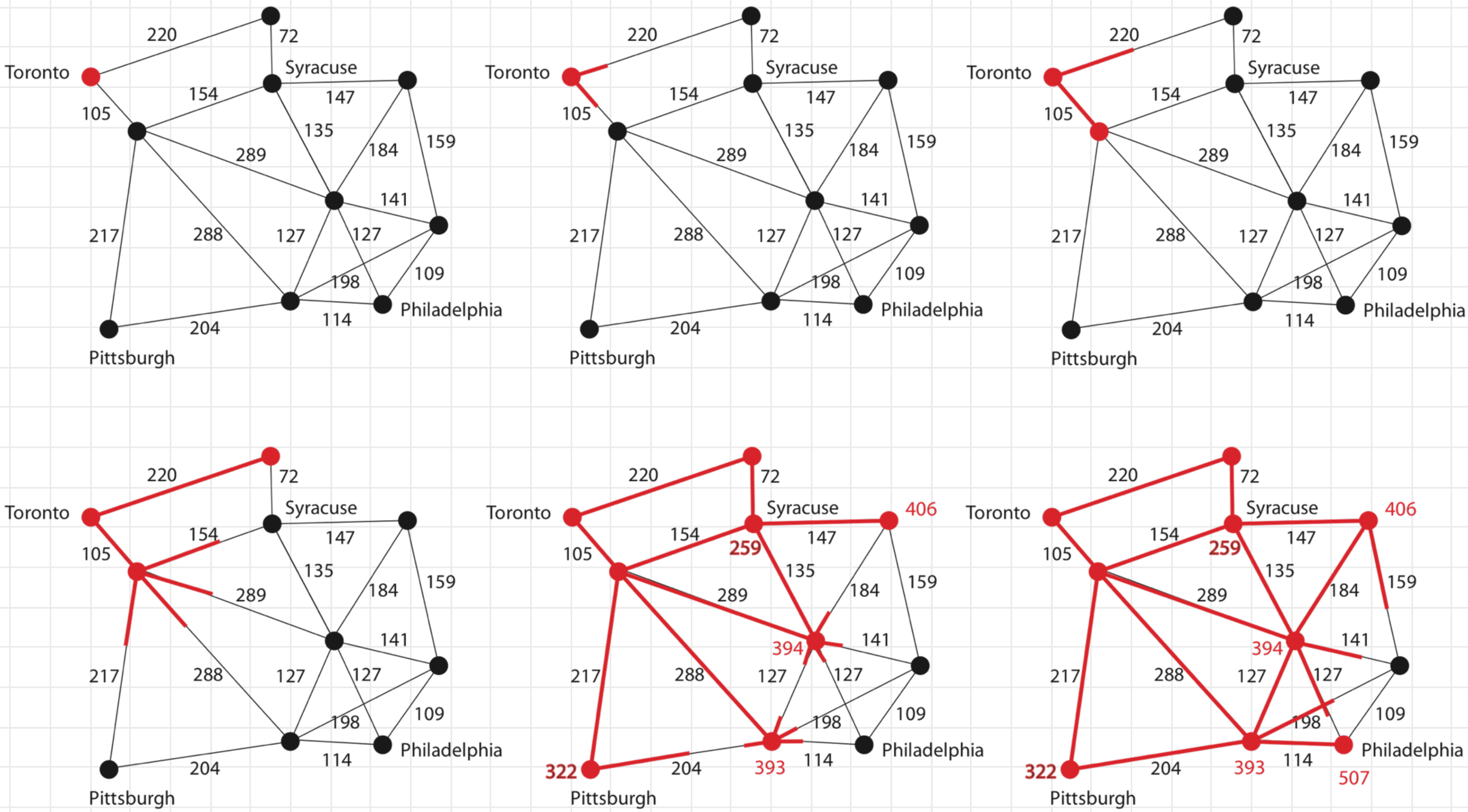
Initialize: S = null d(s) = 0 d(u) = infinity for all other u

update step:

pick v in V - S to minimize d(v)
add v to S
for edge (v, u), u in V - S
d(u) <— min { d(u), d(v) + w(v,u) } (*)

line (*) takes O(m) in total. Use a priority queue to store d(v) values.
Total time O(m + n log n)

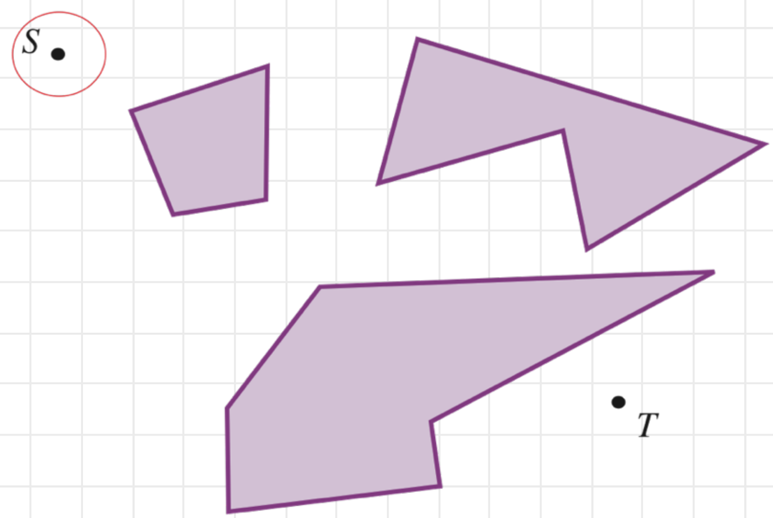
Geometric visualization of Dijkstra's algorithm — imagine paint flowing along edges



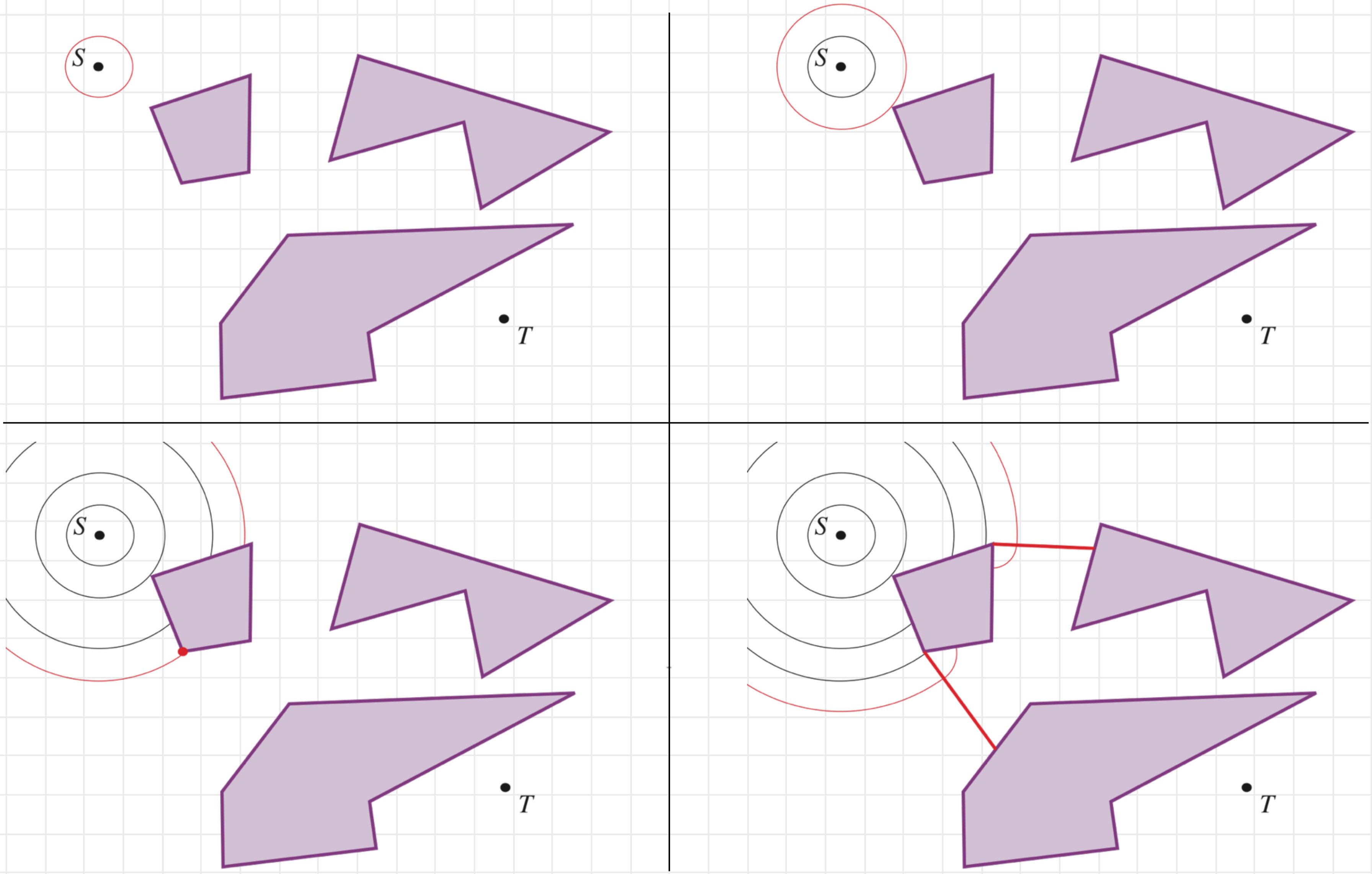
Shortest paths in the plane with polygonal obstacles via continuous Dijkstra

due to Mitchell ’96.
O(n log n), Hershberger, Suri ‘99

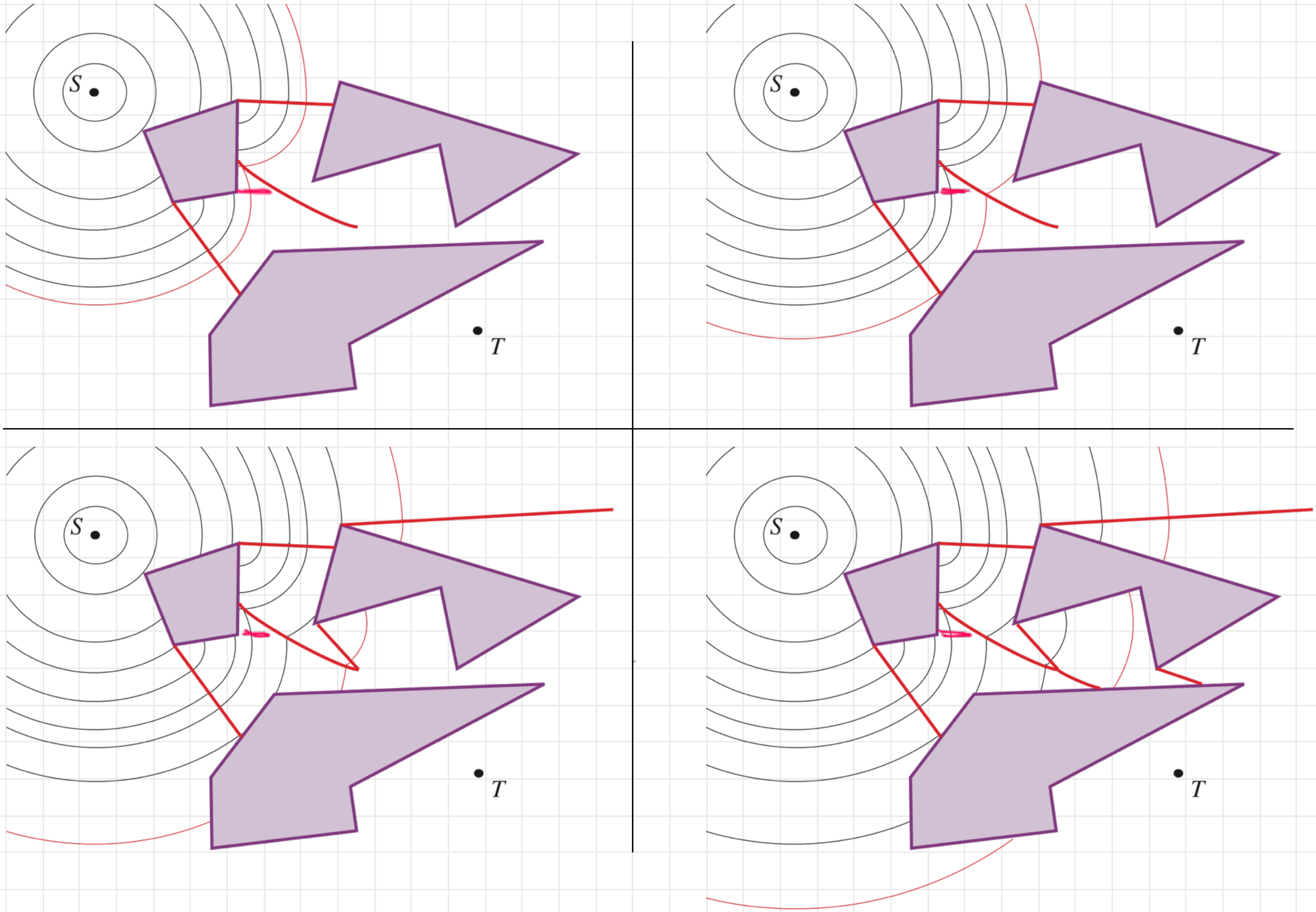
wavefront expands from point S



Continuous Dijkstra approach. Wavefront expands from point S.



Continuous Dijkstra approach. Wavefront expands from point S.



Continuous Dijkstra approach.

Implementation issues:

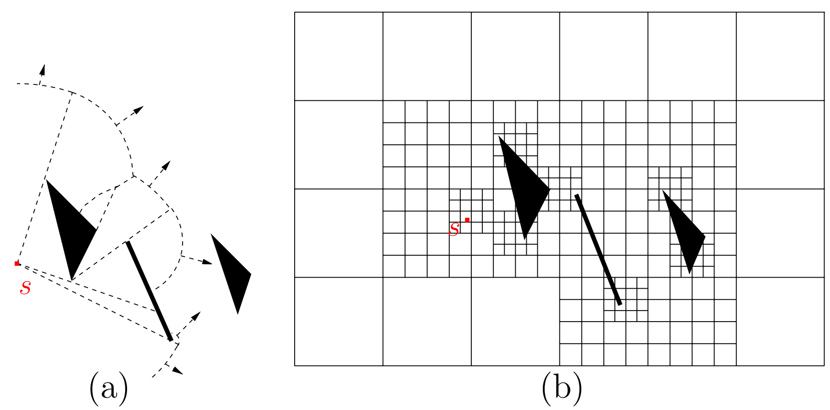
- keep track of events where the wave front changes combinatorially
- find which event occurs next (use a priority queue)
- make updates for that event

Original version was $O(n^2 \log n)$.

Improved to $O(n \log n)$ by Hershberger, Suri

complicated!

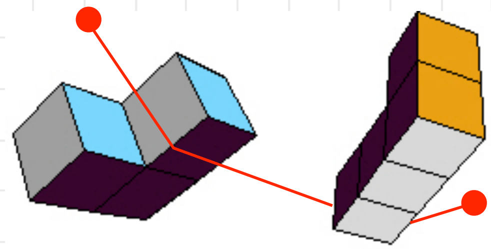
involves subdividing space and approximating the wavefront cell by cell



Schreiber & Sharir

Note: Theta(n log n) is a lower bound

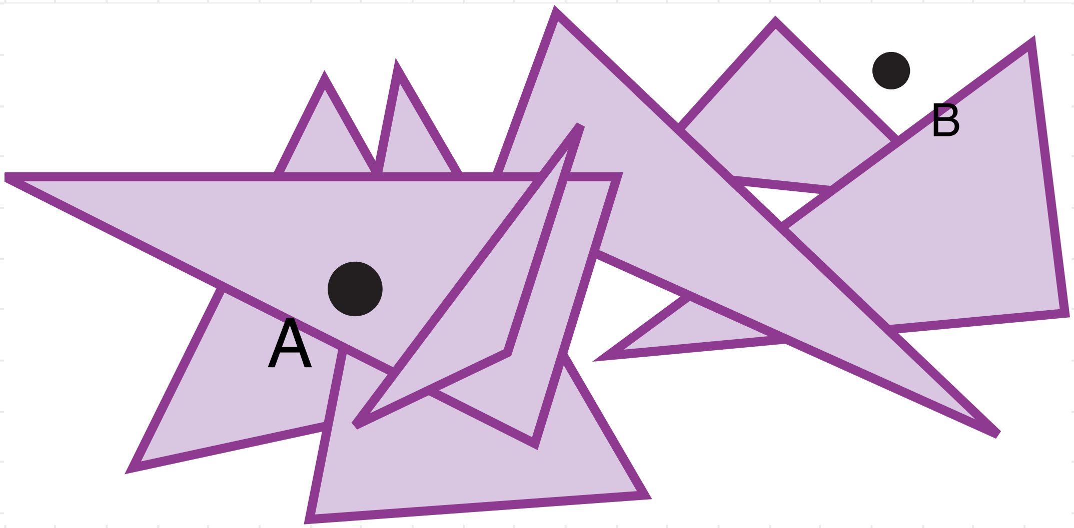
Shortest paths in 3D with polyhedral obstacles



Note that a shortest path does not have to travel on segments between vertices.

Shortest paths in 3D with polyhedral obstacles

NP-hard — Canny & Reif, 1987
even for the case of parallel floating triangles



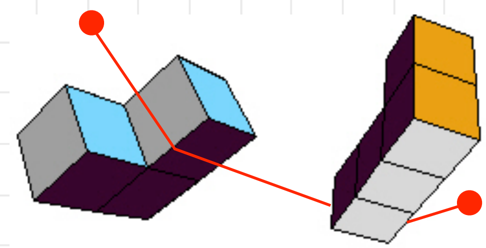
Shortest paths in 3D with polyhedral obstacles

Exact PSPACE algorithm. Canny '88. Uses decidability theory of real closed fields.

Approximation algorithm. Papadimitriou '85;

idea: put many points along each edge and use Dijkstra (on graph)

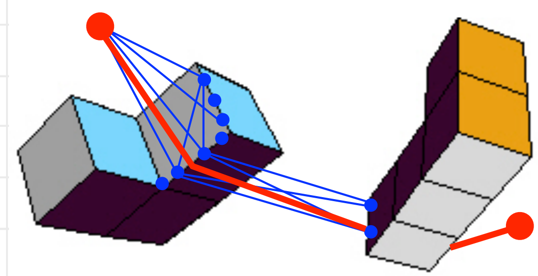
details are a bit tricky:



- points are placed in geometric progression along edges (not placed uniformly)
- this divides the edge into *segments* which become the vertices of the graph

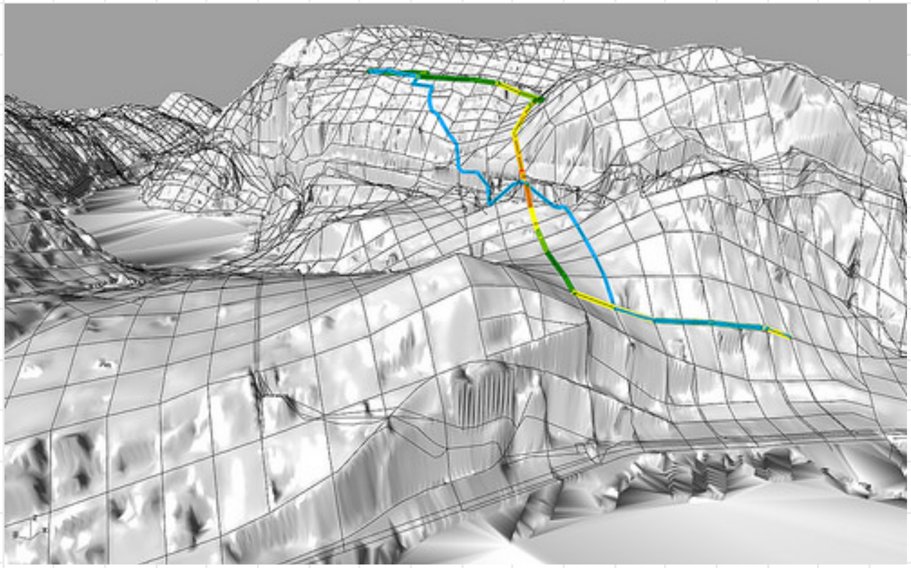
Main Claim. If S and T are at distance *d* then the approximate path has length

$$\leq (1+\epsilon)d$$



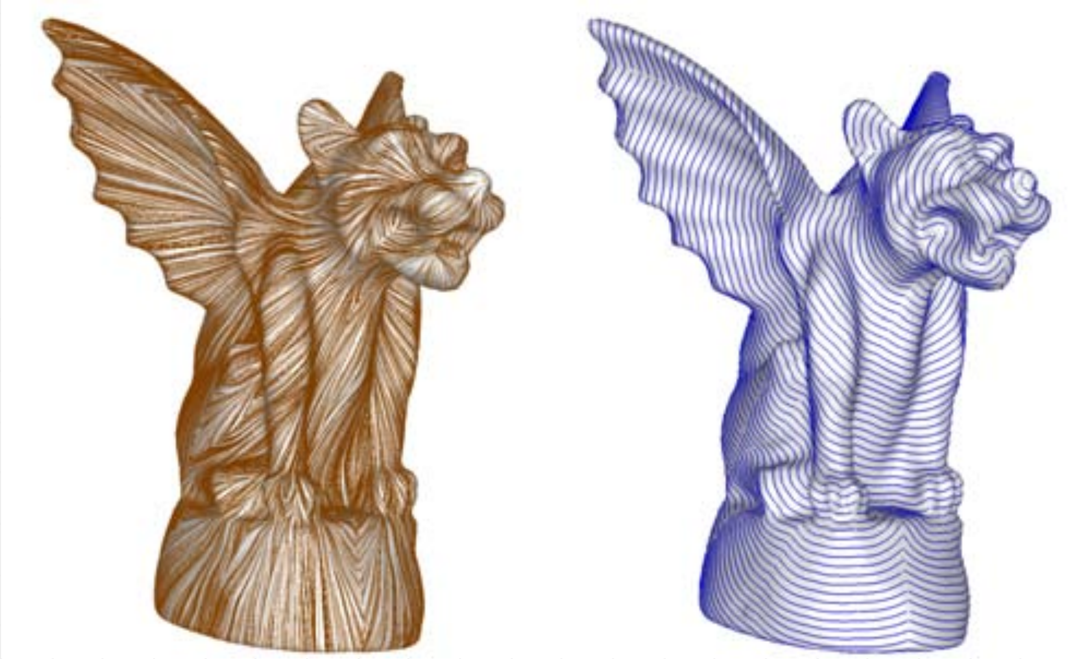
In 2000, Choi, Sellen, Yap, found and corrected an error caused by mixing the algebraic model, where we compute distance using \sqrt, with the bit model used in approximation analysis

Shortest paths on a polyhedral surface



Surface is made up of polygons (usually triangles) joined at edges.
Paths may cut across faces.

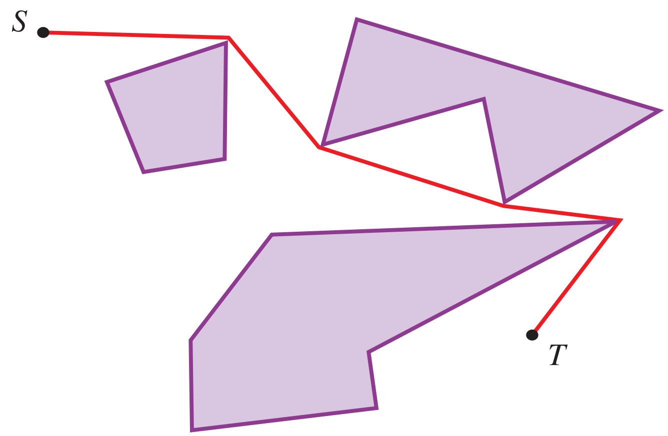
includes shortest paths on surface of polyhedron



Fast Exact and Approximate Geodesics on Meshes
SIGRAPH 2005
https://doi.org/10.1145/1073204.1073228

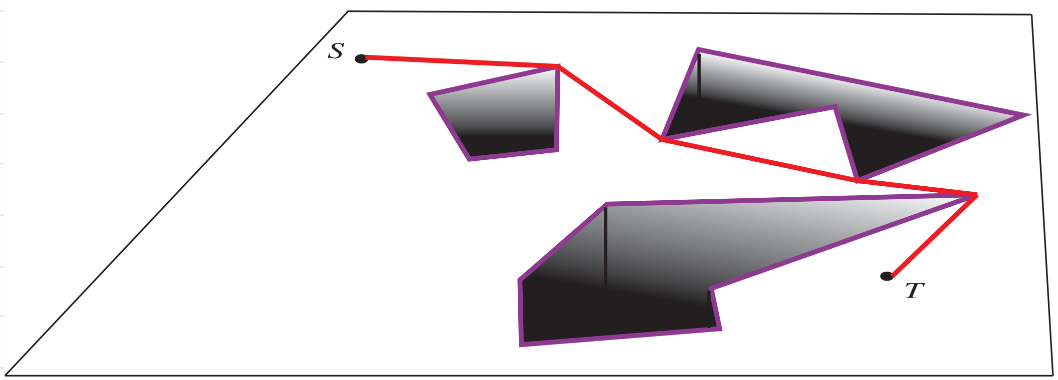
Shortest path on a polyhedral surface

shortest paths among obstacles in the plane	$\subseteq$	shortest paths on a polyhedral surface	$\subseteq$	shortest paths among obstacles in 3D



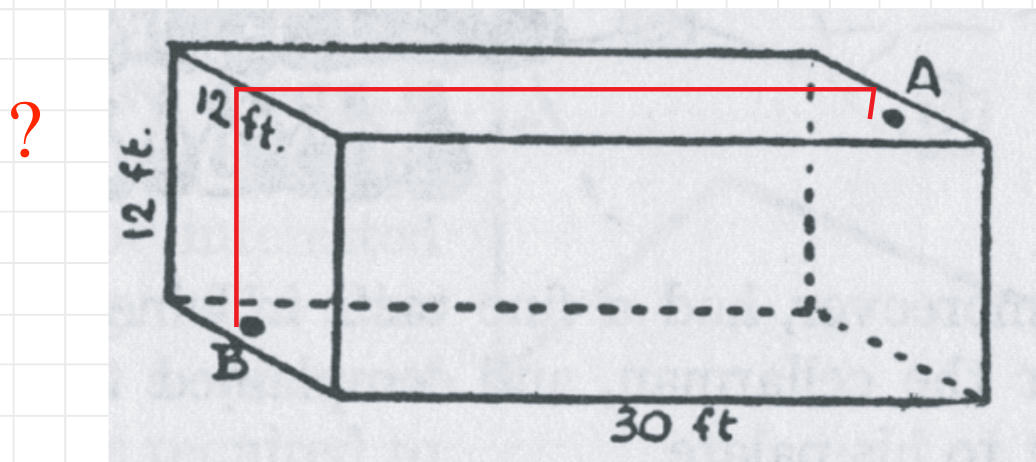
Shortest path on a polyhedral surface

shortest paths among obstacles in the plane	$\subseteq$	shortest paths on a polyhedral surface



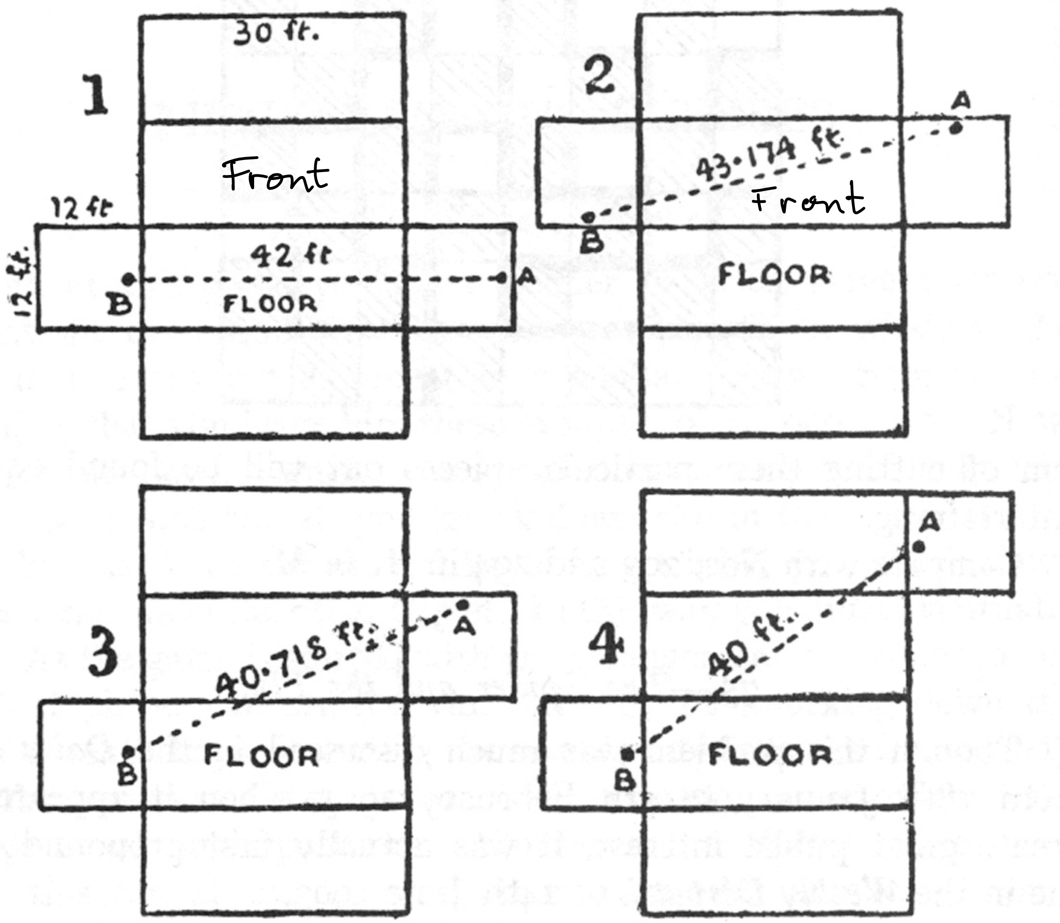
How does a locally shortest path move from one face to another?

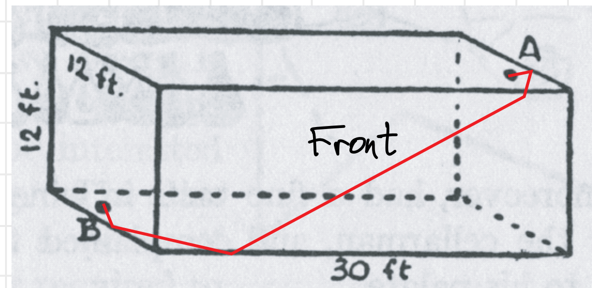
the spider and the fly problem. Dudeney, The Canterbury Puzzles, 1958



How does a locally shortest path move from one face to another?

locally shortest paths are straight lines in unfoldings





Shortest paths on a polyhedral surface

History

$O(n^5)$ O'Rourke and students, '85

$O(n^2 \log n)$ Mitchell, Mount, Papadimitriou, '87 — using continuous Dijkstra approach

$O(n^2)$ Chen and Han, '96

~~$O(n \log^2 n)$ Kapoor '99~~ no longer believed

$O(n \log n)$ for the special case of a convex polyhedron. Schreiber, Sharir, 2006

Chen and Han algorithm to find shortest paths on a polyhedral surface

Input: polyhedral surface made up of triangles in 3-space, joined edge-to-edge (every non-boundary edge is in 2 triangles). Source point s, destination point t. n = # triangles.

First consider a **convex surface**.
Then a shortest path will not go through any vertices.

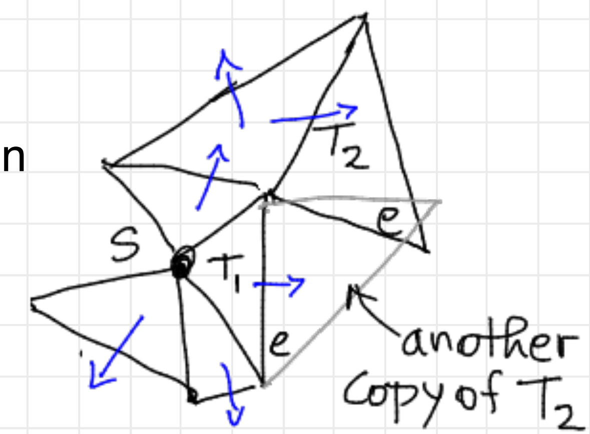
Claim 1. Shortest paths unfold to straight lines.

Claim 2. A shortest path does not enter a face twice (or we could short-cut).

Claim 3. Two shortest s-t paths do not intersect (except at s and t).

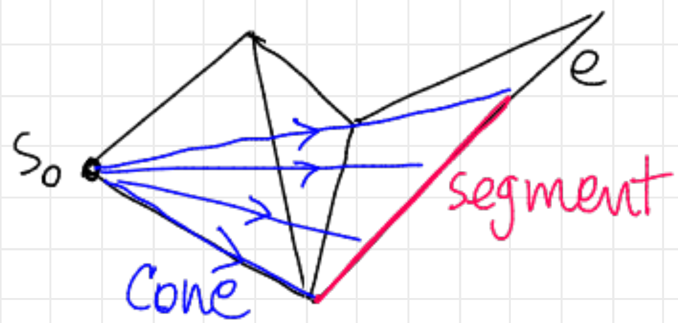
Idea. Start unfolding from the triangle containing s.

- at each edge there is a unique “next” triangle to glue on
- triangles may appear multiple times
- the target t may appear multiple times
- the unfolding will self-overlap in general

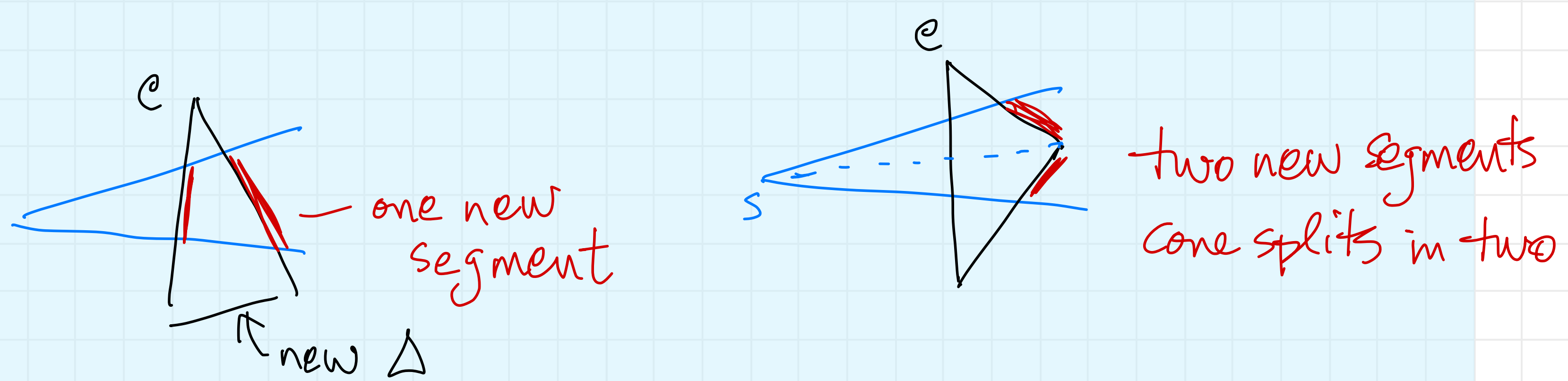


Chen and Han algorithm to find shortest paths on a polyhedral surface

Shortest paths reach a ***segment*** on edge e via a ***cone***.



How a cone expands into the next triangle

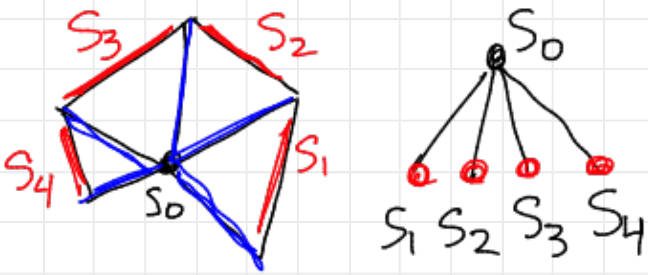


Plan: keep track of segments and of the rays of cones reaching the segment endpoints.

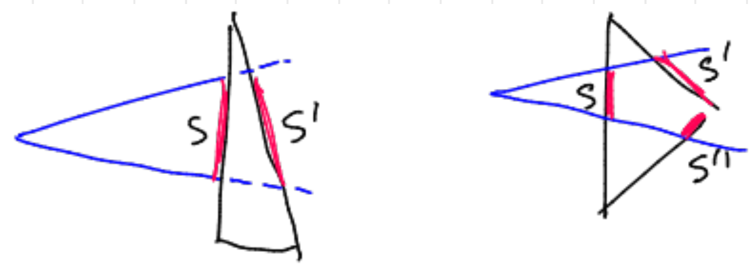
Chen and Han algorithm to find shortest paths on a polyhedral surface

Build a tree. Nodes are the segments.

Initial tree



Each node has one or two children



Lemma. After depth n, the tree contains all shortest paths from s to any point.

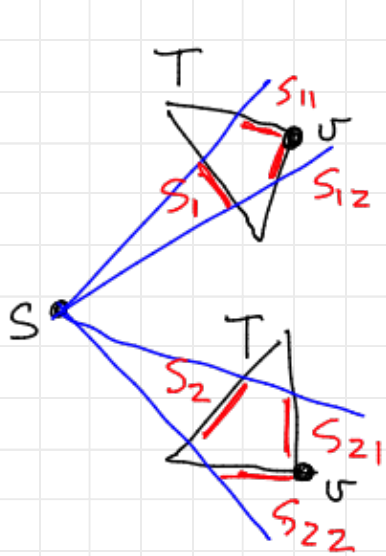
Proof. A shortest path does not repeat a face
So it goes through $\leq n$ faces (= triangles).

Then just compare all straight line paths from s to a copy of t in the tree to find the shortest.

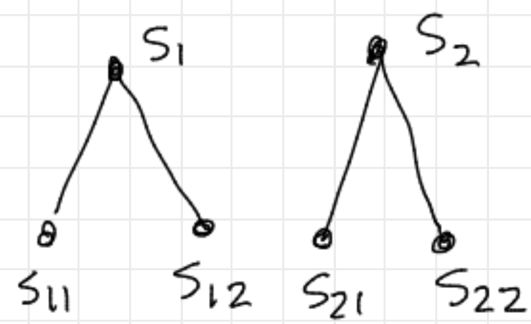
well, ok — but this is exponential size and time!

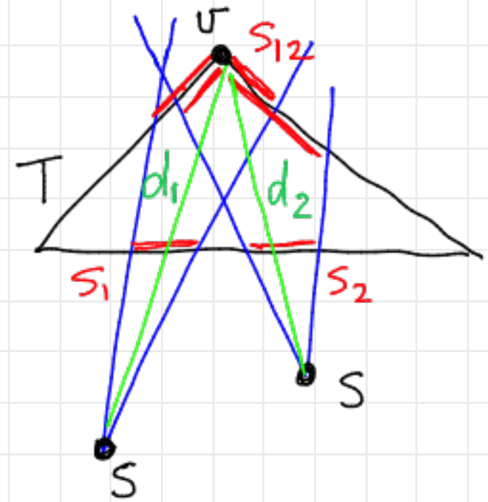
Chen and Han algorithm to find shortest paths on a polyhedral surface

How to prune the segment tree:
Lemma. ("one vertex one cut") Suppose triangle T appears twice and in both cases, a segment splits at vertex v in triangle T. Then we can discard one of the four children.



segments s_1 and s_2 both split at vertex v in copies of triangle T

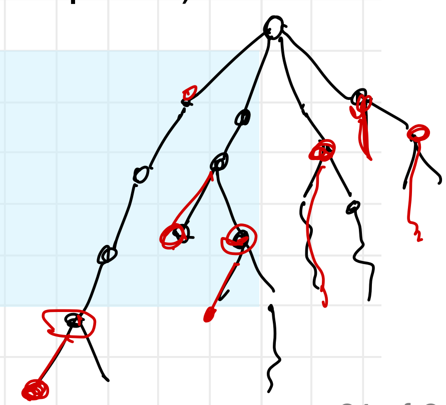




overlaying the copies of T

More precisely, if $d_2 < d_1$ then s_{12} can be discarded (it never gives shortest paths).
Consequence. The size of the segment tree is $O(n^2)$:

There are $O(n)$ leaves in the tree
because the pair v, T ($O(n)$ pairs)
contributes just one branch in the tree



Chen and Han algorithm to find shortest paths on a polyhedral surface

More precisely, if $d_2 < d_1$ then s_{12} can be discarded (it never gives shortest paths).

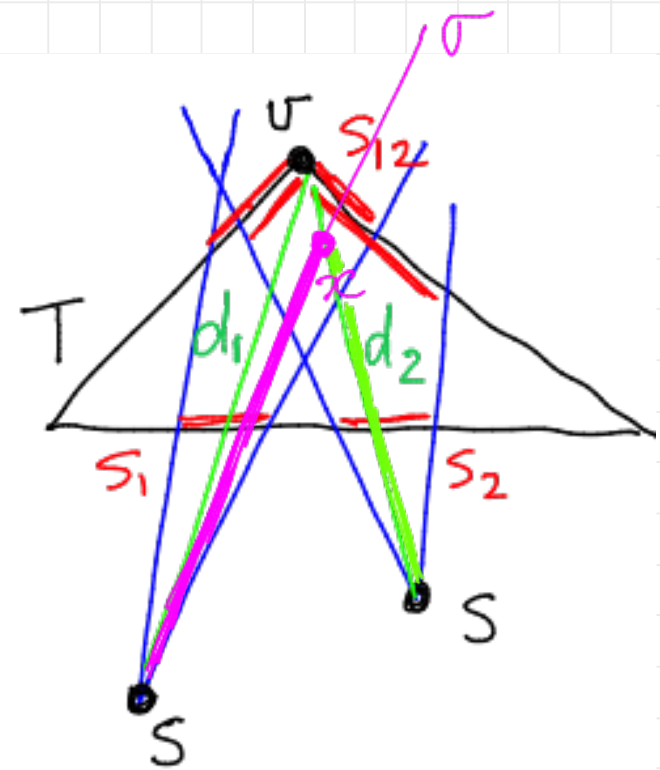
Proof.
Let σ_1 be the path from s to v through segment s_1
Let σ_2 be the path from s to v through segment s_2

Then $|\sigma_1| = d_1$ and $|\sigma_2| = d_2$.

Consider a path σ through s_{12}
σ crosses σ_2 at x

Notation: $\sigma_2(s,x)$ = subpath of σ_2 from s to x

Claim. $|\sigma(s,x)| > |\sigma_2(s,x)|$. Thus s_{12} never gives shortest paths.
Proof.

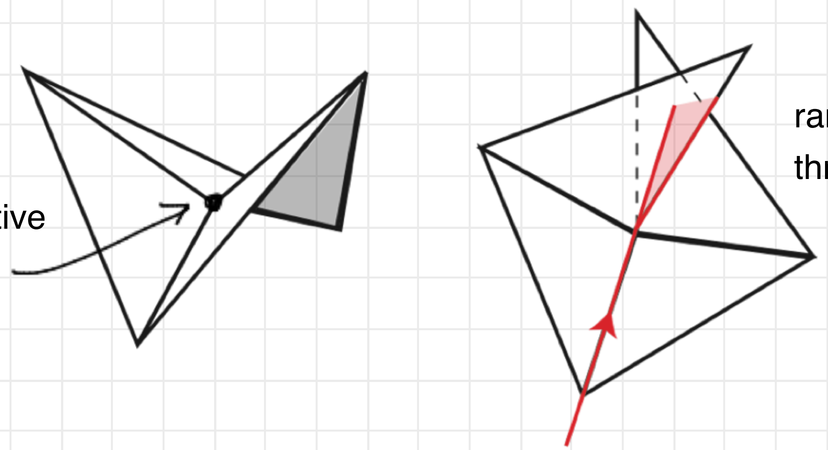


$$|\sigma(s,x)| + |\sigma_2(x,v)|$$
$$> \underbrace{|\sigma_1(s,v)|}_{=d_1} = |\sigma_1| > \overbrace{|\sigma_2|}^{=d_2} \text{ (by assumption)}$$
$$= |\sigma_2(s,x)| + |\sigma_2(x,v)| \quad \boxtimes$$

Chen and Han algorithm to find shortest paths on a polyhedral surface

Dealing with non-convex vertices — actually *negative curvature* vertices.

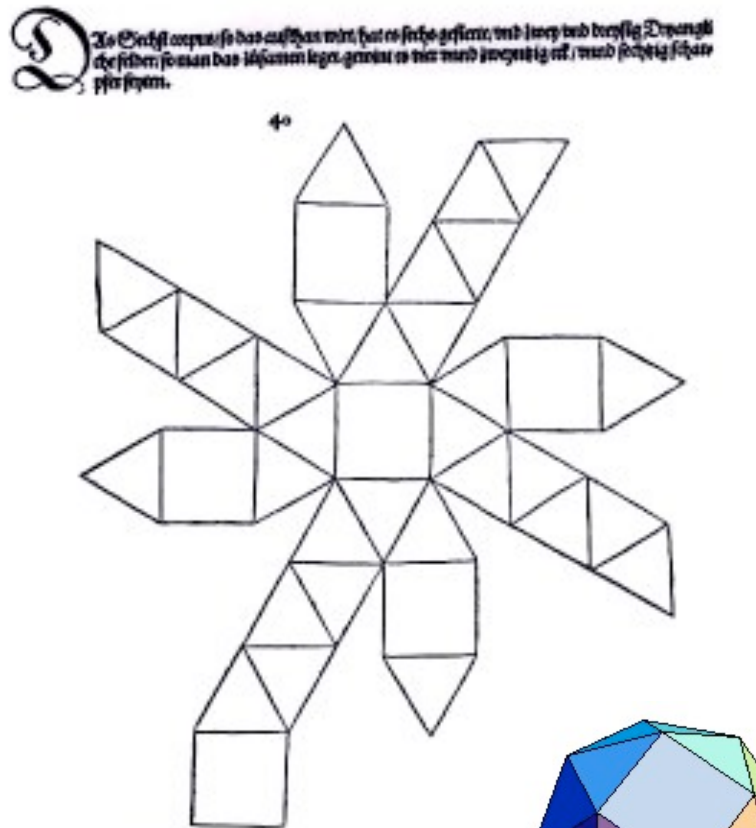
Shortest paths may go through these vertices (think of saddle-points versus mountain tops).

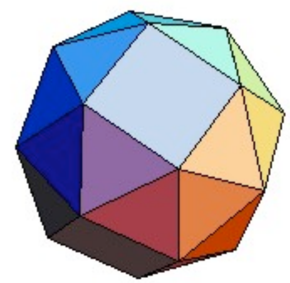


Solution: Each such vertex is treated as a “pseudo-source”.

Application of shortest paths on convex polyhedron: unfolding problem

Durer, 1498



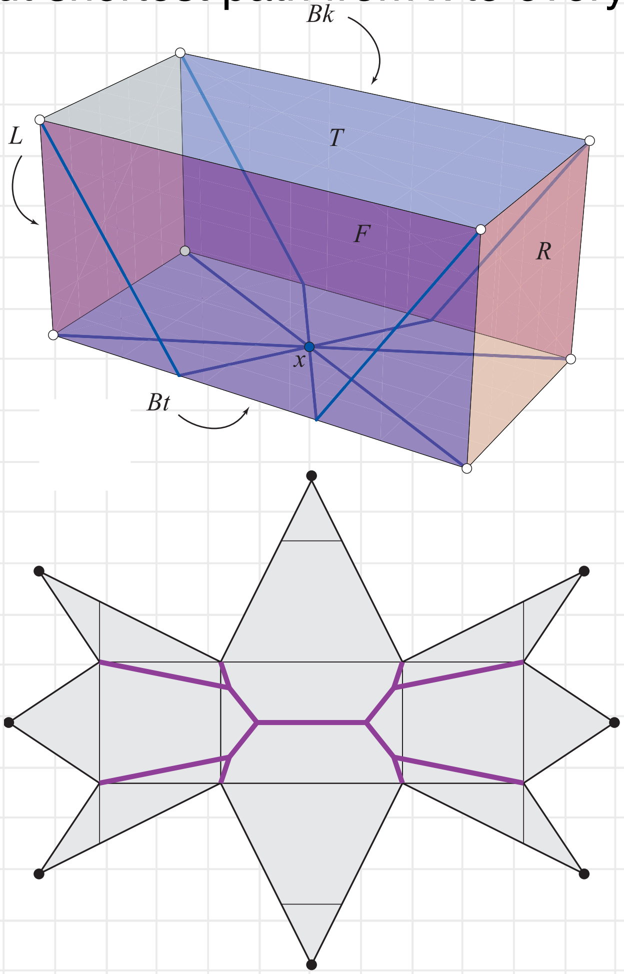


Open problem: can every convex polyhedron be cut on its edges to a planar unfolding?

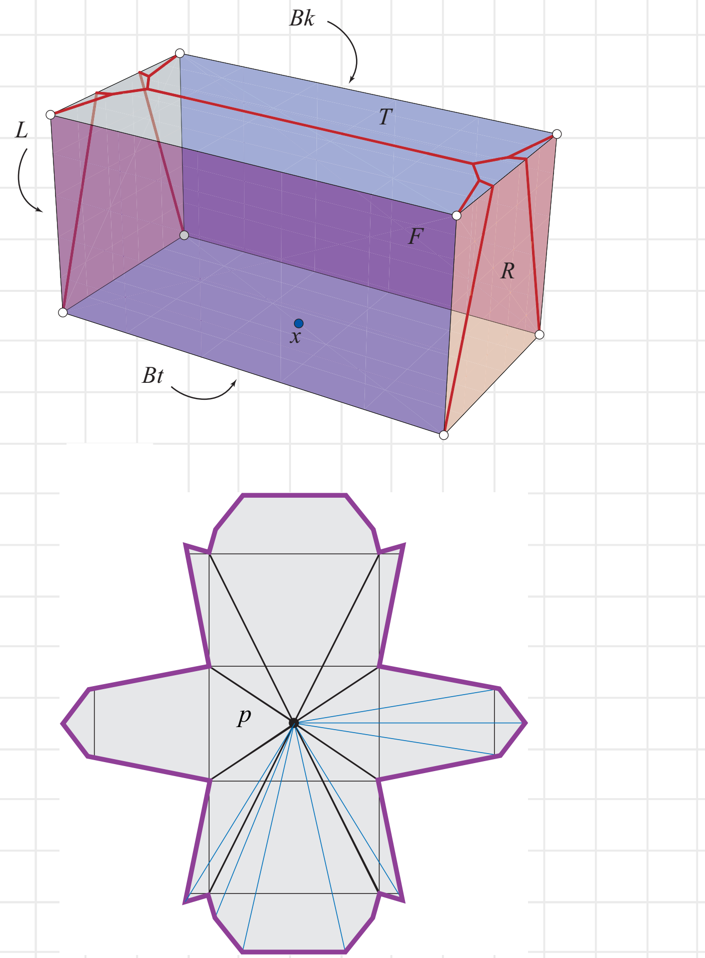
Application of shortest paths on convex polyhedron: unfolding problem

Every convex polyhedron can be unfolded via the source and star unfolding

star unfolding
cut shortest path from x to every vertex



source unfolding
cut Voronoi diagram of x (“ridge tree”)



Summary

- shortest paths:
    - polygons O(n)
    - polygonal domains O(n log n)
    - 3D NP-hard
    - polyhedral surfaces O(n^2)

References

- [CGAA] Chapter 15